```
#!/usr/bin/perl -w

#DESCRIPCIÓN.: Encargado de leer/incoporar los resultados generados por el simulador SST,
crear las estructuras de datos precisas para su procesado y de
#              genrerar gráficas con la métrica obtenidas.
#INVOCACIÓN..: schedlyzer_in <path ficheros .time> <path gráficas a generar>

use strict;

#Módulos necesarios para la generación de gráficas.

use Chart::Clicker;
use Chart::Clicker::Data::Series;
use Chart::Clicker::Data::DataSet;
use Chart::Clicker::Renderer::Bar;

##############################################################################
#Declaración de variables globales.
##############################################################################


my %timesFilesToWork;     #Lista asociativa (hash) de archivos válidos obtenidas del
directorio.

my $pathTimeFiles;
my $pathGraphFiles;
my $timeFileName;         #En cada iteración de bucle se informa con el nombre del fichero
.time procesando.
my $keyAlgorithm;         #En cada iteración de bucle se informa con el algoritmo actualmente
tratado.

#Métricas.

my %CPU_Utilization;
my %throughput;
my %averageTurnaround;
my %averageWaitingTime;

############################################################################## Obtenemos
los ficheros .time a procesar.

##############################################################################
#subrrutina de obtención lista de archivos del directorio.
##############################################################################

sub timeFilesToProcess{

    my @schedulerAlgorithms =("FCFS", "SJF", "EASY");  #Algoritmos que analizaremos.
    my $prefix;
    my $extension;

    opendir(DIRTIMEFILES, $pathTimeFiles) || die "No es posible abrir el directorio: $!";

    while(readdir DIRTIMEFILES){

        #Si el archivo viene prefijado con las siglas de alguno de los algoritmos, lo
        almacenmos.
```

```
        $prefix = substr($_, 0, index($_, "_"));
        $extension = index($_, ".time");

        #if(grep { $_ eq $prefix } @schedulerAlgorithms){
        if($prefix && $extension > 0){
            print "$prefix \n";
            print "$_\n";
            $timesFilesToWork{$prefix} = $_;
        }
    }

    closedir DIRTIMEFILES;
}



################################################################################ Procesamos
cada uno de los ficheros .time.

################################################################################
#Subrrutina para la lectura del contenido del fichero .time
################################################################################

sub processTimeFile{

    my $pPrefix = shift;
    my $pTimeFileName = shift;

    my $lineFile;               #Lectura de cada línea individual.
    my @fields;                 #Obtención de array de campos que conforman la línea.
    my @regsLines;              #Registro de los arrays de campos de cada línea.

    my $totalRun = 0;
    my $totalNOuserJob = 0;
    my $totalJobs = 0;
    my $totalTurnaround = 0;
    my $totalWaitingTime = 0;

    my $maxEndTime = 0;
    my $numProcs = 0;

    my $start = 2;
    my $end = 3;
    my $run = 4;
    my $wait = 5;
    my $resp = 6;
    my $procs = 7;

    my $indRow;
    my $indRow2;
    my $lineCount;

    my $startTime = 0;
    my $theMaxEndTime = 0;

    #my $cont00 = 0;     #Para pruebas de visualización de contenido del array.
```

```
open (TIMEFILE, "$pathTimeFiles/$pTimeFileName") || die "No es posible abrir el fichero:
$!";

while($lineFile = <TIMEFILE>) {
    next if($lineFile =~ /^#/);                 #Saltamos líneas de comentarios.
    next if($lineFile =~ /^\s*$/);              #Saltamos líneas en blanco.
    chop $lineFile;                             #Eliminamos el salto de línea.

    @fields = split /\s+/, $lineFile;

    push (@regsLines, [@fields]);               #Insertamos el array de campos en el array
    de registros de líneas.

    #print $regsLines[$cont00][0] . " " . $regsLines[$cont00][1] . " " .
    $regsLines[$cont00][2] . " " . $regsLines[$cont00][3] . " " . $regsLines[$cont00][4]
    . " " . $regsLines[$cont00][5] . " " . $regsLines[$cont00][6] . " " .
    $regsLines[$cont00][7] . "\n";
    #$cont00++;
}

close TIMEFILE;

##############################################################################
#Odenamos el array bidimensional por orden de ejecución de los procesos.
##############################################################################

@regsLines = sort { $a->[$start] <=> $b->[$start] } @regsLines;   #Alucino con esta
sentencia! :)

#Las visualizamos para comprobar que su ordenación (Descomentar para pruebas).

#foreach $indRow (0..@regsLines-1){
#   print $regsLines[$indRow][0] . " " . $regsLines[$indRow][1] . " " .
$regsLines[$indRow][2] . " " . $regsLines[$indRow][3] . " " . $regsLines[$indRow][4] . "
" . $regsLines[$indRow][5] . " " . $regsLines[$indRow][6] . " " . $regsLines[$indRow][7]
. "\n";
#}

##############################################################################
#Obtención de métricas.
##############################################################################

#Inicializamos totalizadores.

$totalRun = 0;
$totalNOuserJob = 0;
$totalJobs = 0;
$totalTurnaround = 0;
$totalWaitingTime = 0;

foreach $indRow (0..@regsLines-1){

    #Calculamos algunos totales de tiempo.

    $totalRun += $regsLines[$indRow][$run];
    $totalTurnaround += $regsLines[$indRow][$resp];
```

```
        $totalWaitingTime += $regsLines[$indRow][$wait];

        if($indRow>0){

            #Si el timpo de inicio del siguiente job es superior al final del anterior,
            debemos calcular un intervalo de
            #tiempo que el procesador ha estado ejecutando código no de usuario.

            #Obtenemos el tiempo mayor de ejecución desde la entrada del siguiene job hacia
            atrás, para saber si existe algún proceso en máquina y
            #por tanto el sistema estaba ejecutando un proceso de usuario.

            $maxEndTime = 0;
            $lineCount = 0;

            $indRow2 = $indRow-1;

            while($indRow2 >=0 && $lineCount < 500){
                $maxEndTime = $regsLines[$indRow2][$end] > $maxEndTime? $regsLines[$indRow2][
                $end]: $maxEndTime;

                $indRow2--;
                $lineCount++;
            }

            #Si el tiempo de inicio del proceso es > que el tempo de end maximo de los
            procesos que anteteriomente pasaron a ejecutarse
            #es que ha habido un espacio de tiempo en el que el sistema no ejecutaba
            procesos de usuario.

            if($regsLines[$indRow][$start] > $maxEndTime){
                $totalNOuserJob +=  $regsLines[$indRow][$start] - $maxEndTime;
            }
        }

        #Necesario para calcular el intervalo de tiempo de ejecución total.

        $startTime = $regsLines[0][$start];
        $theMaxEndTime = $regsLines[$totalJobs][$end] > $theMaxEndTime? $regsLines[$totalJobs
        ][$end]: $theMaxEndTime;

        $totalJobs++;

    }

    #Obtenemos el porcentaje de las métricas:

    print "\n $pPrefix \n";

    ##############################################################% uso del procesador.

    $CPU_Utilization{$pPrefix} = ($totalRun / ($totalRun + $totalNOuserJob))*100;

    print $totalRun . " Total run \n";
    print $totalNOuserJob . " Total no user job \n";

    ############################################################## Throughput
```

```
    (Productividad).

    print "startTime = " . $startTime . "\n";
    print "theMaxEndTime = " . $theMaxEndTime . "\n";

    $throughput{$pPrefix} = $totalJobs / ((($theMaxEndTime - $startTime))/86400);
    #Convertimos de 'segundos' a 'días'.

    print $totalJobs . " Total jobs \n";

    ################################################################ Turnaround time (Tiempo
    de retorno).

    $averageTurnaround{$pPrefix} = ($totalTurnaround / $totalJobs) / 3600;
    #Convertimos de 'segundos' a 'horas'.

    print $totalTurnaround . " Total Turnaround \n";

    ################################################################ Waiting time (Tiempo de
    espera).

    $averageWaitingTime{$pPrefix} = ($totalWaitingTime / $totalJobs) / 3600;
    #Convertimos de 'segundos' a 'horas'

    print $totalWaitingTime . " totalWaitingTime. \n";


    print "************************************************************** \n";

}
################################################################################ Generación
de gráfica por cada métrica a analizar.

##############################################################################
#Generación de gráfica.
##############################################################################

sub generatesGraphic{

    my $chart;
    my $dataset;

    #Por cada métrica a analizar debemos recorrer la tabla hash de cada una de ellas para
    obtener el valor de cada algoritmo.

    ##############################################################################
    $CPU_Utilization

    # build the chart
    $chart = Chart::Clicker->new(width => 800, height => 500);

    # Create an empty dataset that we can add to
    $dataset = Chart::Clicker::Data::DataSet->new;


    foreach $keyAlgorithm (sort keys %timesFilesToWork){
```

```
    $dataset->add_to_series(Chart::Clicker::Data::Series->new(
        keys    => [ 1 ],
        values  => [ $CPU_Utilization{$keyAlgorithm} ],
        name    => $keyAlgorithm . ": " . sprintf "%.3f", $CPU_Utilization{$keyAlgorithm},
    ));

    #print $keyAlgorithm. "\n";
    #print $CPU_Utilization{$keyAlgorithm}. "\n";
}

outputGraphic($chart, $dataset, 'CPU Utilization', '%', '%.3f');

########################################################################## Throughput
(Productividad).

# build the chart
$chart = Chart::Clicker->new(width => 800, height => 500);

# Create an empty dataset that we can add to
$dataset = Chart::Clicker::Data::DataSet->new;

foreach $keyAlgorithm (sort keys %timesFilesToWork){

    $dataset->add_to_series(Chart::Clicker::Data::Series->new(
        keys    => [ 1 ],
        values  => [ $throughput{$keyAlgorithm} ],
        name    => $keyAlgorithm . ": " . sprintf "%.2f", $throughput{$keyAlgorithm},
    ));

    #print $keyAlgorithm. "\n";
    #print $throughput{$keyAlgorithm}. "\n";
}

outputGraphic($chart, $dataset, 'Throughput', 'jobs/day', '%.2f');

################################################################ Turnaround time (Tiempo
de retorno).

# build the chart
$chart = Chart::Clicker->new(width => 800, height => 500);

# Create an empty dataset that we can add to
$dataset = Chart::Clicker::Data::DataSet->new;

foreach $keyAlgorithm (sort keys %timesFilesToWork){

    $dataset->add_to_series(Chart::Clicker::Data::Series->new(
        keys    => [ 1 ],
        values  => [ $averageTurnaround{$keyAlgorithm} ],
        name    => $keyAlgorithm . ": " . sprintf "%.2f", $averageTurnaround{$keyAlgorithm
        },
    ));

    #print $keyAlgorithm. "\n";
    #print $averageTurnaround{$keyAlgorithm}. "\n";
}
```

```
    outputGraphic($chart, $dataset, 'Average Turnaround', 'Hours', '%.2f');

    ################################################################ Waiting time (Tiempo de
    espera).

    # build the chart
    $chart = Chart::Clicker->new(width => 800, height => 500);

    # Create an empty dataset that we can add to
    $dataset = Chart::Clicker::Data::DataSet->new;

    foreach $keyAlgorithm (sort keys %timesFilesToWork){

        $dataset->add_to_series(Chart::Clicker::Data::Series->new(
            keys    => [ 1 ],
            values  => [ $averageWaitingTime{$keyAlgorithm} ],
            name    => $keyAlgorithm . ": " . sprintf "%.2f", $averageWaitingTime{
            $keyAlgorithm},
        ));

        #print $keyAlgorithm. "\n";
        #print $averageWaitingTime{$keyAlgorithm}. "\n";
    }

    outputGraphic($chart, $dataset, 'Average Waiting Time', 'Hours', '%.2f');

}

sub outputGraphic{

    my $chart = shift;
    my $dataset = shift;
    my $pMetric = shift;
    my $pYLabel = shift;
    my $pFormat = shift;

    #Título del chart.
    $chart->title->text($pMetric);

    # add the dataset to the chart
    $chart->add_to_datasets($dataset);

    my $context = $chart->get_context('default');
    $context->range_axis->format($pFormat);
    $context->range_axis->label($pYLabel);
    $context->domain_axis->hidden(1);
    $context->domain_axis->label('Job scheduling policies');

    my $area = Chart::Clicker::Renderer::Bar->new(opacity => .6);
    $area->brush->width(2);
    $context->renderer($area);

    # write the chart to a file
    $chart->write_output("$pathGraphFiles/$pMetric.png");
}

############################################################################ Main
```

```
#En la invocación de este script debemos esperar dos parámetros de entrada:
#1: Directorio de donde obtner los ficheros .time a procesar.
#2: Directorio donde dejar las gráficas generadas.

if($#ARGV > 0 && $#ARGV <3){    #Si nos han pasado exactamente dos parámetros

    #Inicializamos variables de origen de fichero .time y destino de gráficas generadas.

    $pathTimeFiles = $ARGV[0];
    $pathGraphFiles = $ARGV[1];

    timeFilesToProcess();

    #Recorremos la tabla hash de ficheros a tratar. Por cada fichero (algoritmo)
    calcularemos las métricas a analizar y genermos su gráfica.

    while (($keyAlgorithm, $timeFileName) = each(%timesFilesToWork)){
        processTimeFile($keyAlgorithm, $timeFileName);

    }

    generatesGraphic();
}
else{
    print "Se han recibido $#ARGV parámetros. El número de parámetros no es correcto. \n";
}

exit;
```